

## DS4 : Graphes de flot

### Partie 0 Présentation des graphes de flot

Cette partie ne comporte aucune question. Elle introduit le problème étudié dans tout le sujet ainsi que le vocabulaire permettant de le formuler. La lire attentivement et ne pas hésiter à s'y référer au fil de l'énoncé. L'exemple introduit dans cette partie servira dans les suivantes.

Un *graphe de flot* est un quintuplet  $G = (S, A, c, s, t)$  où :

- $(S, A)$  est un graphe orienté et sans boucle.
- $s \in S$  est une source de  $(S, A)$  c'est-à-dire un sommet de degré entrant nul et  $t$  est un puits de  $(S, A)$  c'est à dire un sommet de degré sortant nul.
- $c$  est une fonction de  $A$  dans  $\mathbb{R}^+$  appelée *fonction de capacité*.

La fonction de capacité  $c$  est étendue des arcs à l'ensemble  $S^2$  en posant  $c(u, v) = 0$  si  $(u, v) \notin A$ . Par exemple, le graphe  $G_0$  est un graphe de flot dans lequel on a représenté les capacités directement sur les arcs de manière similaire à la façon dont on indique les poids dans un graphe pondéré :

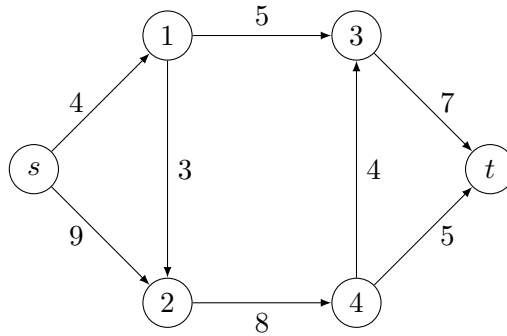


FIGURE 1 - Le graphe de flot  $G_0$ .

Si  $G = (S, A, c, s, t)$  est un graphe de flot, on appelle *flot* sur  $G$  une fonction  $f : S^2 \rightarrow \mathbb{R}$  vérifiant les trois propriétés suivantes :

- Pour tous  $u, v \in S$ ,  $f(u, v) = -f(v, u)$  (antisymétrie).
- Pour tous  $u, v \in S$ ,  $f(u, v) \leq c(u, v)$  (respect de la capacité).
- Pour tout  $u \in S \setminus \{s, t\}$ ,  $\sum_{v \in S} f(u, v) = 0$  (conservation).

On appelle *débit du flot*  $f$  la quantité  $|f| = \sum_{u \in S} f(s, u)$ . Par exemple, on représente en figure 2 un flot sur le graphe  $G_0$  de la façon suivante : sur chaque arc  $(u, v)$  on indique, séparés par un symbole /, la valeur de  $f(u, v)$  et la valeur de  $c(u, v)$ . On n'indique les valeurs de  $f(u, v)$  que pour  $(u, v) \in A$ , les valeurs de  $f(u, v)$  pour  $(u, v) \notin A$  se déduisant des trois propriétés caractérisant un flot.

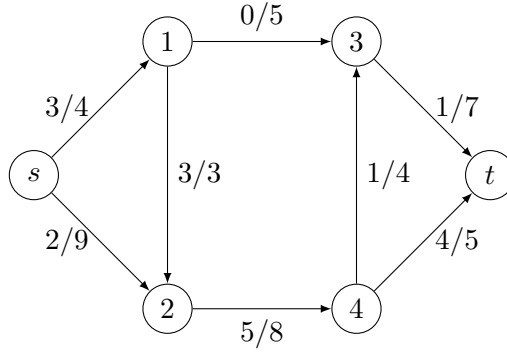


FIGURE 2 - Le graphe de flot  $G_0$  et un flot  $f$ .

Par ailleurs, si  $f$  est un flot sur un graphe de flot et  $u$  est un sommet de ce graphe, on appelle :

- *flux sortant de  $u$*  la quantité  $\varphi_+(u) = \sum_{v \in S, f(u,v) > 0} f(u,v)$ .
- *flux entrant de  $u$*  la quantité  $\varphi_-(u) = \sum_{v \in S, f(u,v) < 0} f(v,u)$ .
- *flux net de  $u$*  la quantité  $\varphi(u) = \varphi_+(u) - \varphi_-(u)$

L'objectif de ce sujet est d'étudier le problème MAXFLOW défini comme suit :

**Entrée :** Un graphe de flot  $G$ .

**Sortie :** Un flot  $f$  maximal sur  $G$  c'est-à-dire un flot sur  $G$  tel que son débit  $|f|$  soit maximal.

Le calcul de flots maximaux permet la résolution de nombreux problèmes d'optimisation de la circulation de flux dans un réseau : voitures sur des routes, paquets de données sur Internet, électricité dans des câbles... Dans chacun de ces cas, l'objectif est d'acheminer la plus grande quantité de données / matière entre la source  $s$  et la destination  $t$  tout en respectant les contraintes physiques du réseau.

### Partie 1 Quelques propriétés d'un flot

1. Dans le contexte du flot  $f$  décrit par la figure 2, indiquer en justifiant brièvement :
  - a) La valeur du flot pour les couples de sommets  $(1,2)$ ,  $(3,4)$  et  $(1,4)$ .
  - b) Le débit du flot  $f$ .
  - c) Le flux entrant du sommet  $t$ , le flux sortant du sommet 2 et le flux net du sommet 3.

Pour les questions 2, 3, et 4,  $G = (S, A, c, s, t)$  désigne un graphe de flot quelconque et  $f$  un flot sur  $G$ .

2. Justifier que  $f(u,v) > 0$  implique que  $(u,v) \in A$  et que  $f(u,v) < 0$  implique  $(v,u) \in A$ .
3. Montrer que si  $f$  est un flot alors la propriété suivante est vraie :

$$\forall u \in S \setminus \{s, t\}, \varphi(u) = 0$$

Expliquer en français ce à quoi correspond cette égalité pour le sommet  $u$ .

4. Montrer que  $|f| = \varphi(s) = -\varphi(t)$ .
5. Représenter graphiquement une solution à MAXFLOW pour le graphe  $G_0$  défini à la figure 1 en justifiant que le flot proposé est bien maximal.

## Partie 2 Algorithme de Ford-Fulkerson

Dans cette partie et les suivantes, sauf indication contraire,  $G = (S, A, c, s, t)$  désigne un graphe de flot quelconque et  $f$  un flot sur  $G$ . Pour déterminer un flot maximal dans  $G$ , on se munit d'une opération de *saturation de chemin* dont la définition suit :

- La *capacité disponible* d'un arc  $(u, v) \in A$  est la quantité  $c(u, v) - f(u, v)$ . Un arc est dit *saturé* si sa capacité disponible est nulle.
- La *capacité disponible* d'un chemin de  $s$  à  $t$  dans  $G$  est le minimum des capacités disponibles des arcs de ce chemin. Un chemin de  $s$  à  $t$  est dit saturé si sa capacité disponible est nulle.
- Si  $C$  est un chemin de  $s$  à  $t$  dans  $G$  qui dispose d'une capacité disponible  $m > 0$ , la *saturation* du chemin  $C$  consiste à modifier le flot  $f$  de sorte à ce que, pour tout arc  $(u, v)$  du chemin  $C$ ,  $f(u, v)$  est remplacé par  $f(u, v) + m$  et  $f(v, u)$  est remplacé par  $f(v, u) - m$ .

Par exemple, la saturation du chemin  $(s, 2, 4, 3, t)$  dans le graphe  $G_0$  à partir du flot défini en figure 2 modifie ce dernier en augmentant son débit de 3 comme suit :

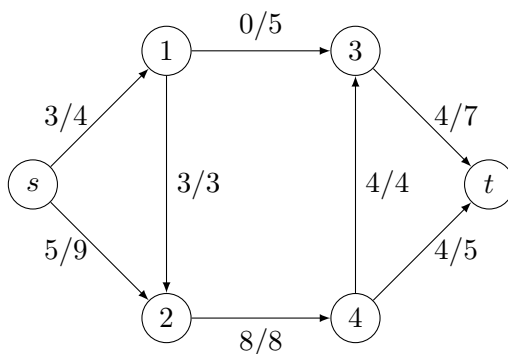


FIGURE 3 - Le flot de la figure 2 après saturation du chemin  $(s, 2, 4, 3, t)$

On considère alors l'algorithme `flot_glouton` suivant :

**Entrée :** Un graphe de flot  $G = (S, A, c, s, t)$   
 Pour tout  $(u, v) \in S^2$ ,  $f(u, v) \leftarrow 0$   
 Tant qu'il existe un chemin  $C$  non saturé de  $s$  à  $t$   
     Saturer  $C$   
 Renvoyer  $f$

6. Expliquer comment trouver un chemin non saturé de  $s$  à  $t$  dans un graphe de flot, étant donnés ce graphe et le flot  $f$ , et déterminer le coût d'une telle opération.
7. Montrer qu'après saturation d'un chemin  $C$ ,  $f$  est toujours un flot sur  $G$  et que  $C$  est saturé. En déduire que l'algorithme `flot_glouton` termine et renvoie un flot sur  $G$ .
8. Appliquer l'algorithme `flot_glouton` au graphe  $G_0$  à partir du flot représenté en figure 3 et représenter graphiquement le résultat. Le flot obtenu est-il maximal ?

Afin de corriger ce problème, il faut autoriser à faire "refluer" le flot en arrière le long d'un arc. On définit le *graphe résiduel associé* à  $f$  par  $G_f = (S, A_f)$  où  $A_f = \{(u, v) \in S^2 \mid c(u, v) - f(u, v) > 0\}$ . Un chemin de  $s$  à  $t$  dans  $G_f$  est appelé *chemin améliorant pour  $f$* . Par exemple, le graphe résiduel associé au flot  $f$  défini en figure 3 est :

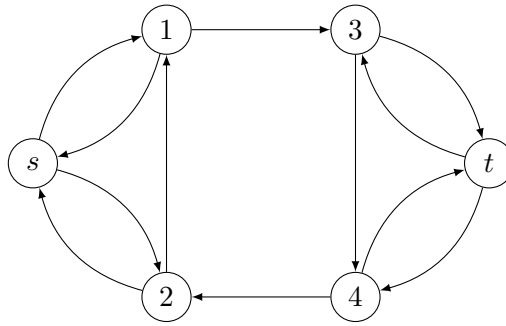


FIGURE 4 - Le graphe résiduel associé au flot  $f$  sur  $G_0$  présenté à la figure 3.

Remarquez en particulier que le graphe résiduel  $G_f$  peut contenir des arcs qui n'étaient pas dans le graphe  $G$  initial. On prendra garde à ce fait dans la suite.

9. Montrer que si  $(u, v) \in A_f$  alors  $(u, v) \in A$  ou  $(v, u) \in A$ .

10. Représenter graphiquement le graphe résiduel de  $G_0$  associé au flot obtenu à la question 8.

L'algorithme de Ford-Fulkerson est le suivant :

**Entrée :** Un graphe de flot  $G = (S, A, c, s, t)$   
**Sortie :** Un flot maximal  $f$  sur  $G$   
 Pour tout  $(u, v) \in A$ ,  $f(u, v) \leftarrow 0$   
 Tant qu'il existe un chemin améliorant pour  $f$   
     Saturer ce chemin  
 Renvoyer  $f$

11. En initialisant le flot par les valeurs trouvées à la questions 8 plutôt que par 0, appliquer l'algorithme de Ford-Fulkerson au graphe  $G_0$ . Le flot obtenu est-il maximal ?

### Partie 3 Implémentation de l'algorithme de Ford-Fulkerson

Dans cette partie, on implémente l'algorithme de Ford-Fulkerson en C. On suppose avoir chargé les bibliothèques `stdio`, `stdlib`, `stdint`, `stdbool`, `assert` ainsi que la bibliothèque `limits` qui fournit entre autres deux constantes `INT_MAX` et `INT_MIN`. On suppose disposer également de deux fonctions de prototypes `int min(int x, int y)` et `int max(int x, int y)` dont les spécifications devraient être évidentes.

On cherche tout d'abord à implémenter une structure de file dotée des opérations suivantes :

```

//création d'une file vide, complexité O(1)
file* creer_file(void);
//libération de la mémoire, complexité O(longueur)
void liberer_file(file* f);
// nombre d'éléments présents dans la file, complexité O(1)
int longueur_file(file* f);
// extraction du plus ancien élément (erreur si vide), complexité O(1)
int extraire_file(file* f);
// ajout d'un élément, complexité O(1)
void ajouter_file(file* f, int v);
  
```

On construit cette structure à l'aide d'une liste simplement chaînée en conservant un pointeur sur le premier élément de la file et un sur le dernier, ce qui conduit à la définition des structures suivantes :

```

struct maillon{
    int donnee;
    struct maillon* suivant;
};
typedef struct maillon maillon;

struct file{
    maillon* debut;
    maillon* fin;
    int longueur;
};
typedef struct file file;

```

Dans la structure `file`, le champ `debut` pointe vers le maillon le plus ancien (qui sera donc la tête de la file), le champ `fin` pointe vers le maillon le plus récent (donc, le dernier élément de la file) et le champ `longueur` contient le nombre d'éléments actuellement présents dans la file. On veillera à maintenir l'invariant suivant : si la file est vide, `longueur` vaut 0 et les champs `debut` et `fin` valent `NULL`.

On illustre ci-dessous l'insertion de 8 dans la file (4,3,1,2) suivie de l'extraction de la tête de la file obtenue. Les éléments modifiés apparaissent en gras et ceux supprimés en pointillés.

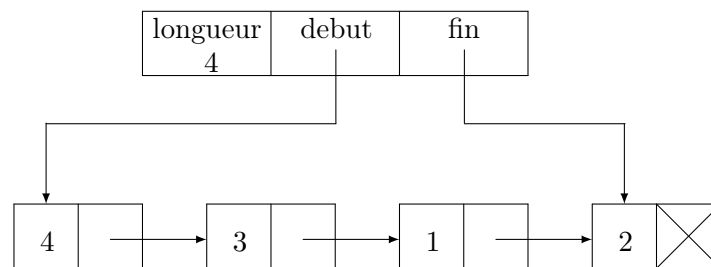


FIGURE 5 - La file  $\leftarrow (4, 3, 1, 2) \leftarrow$ .

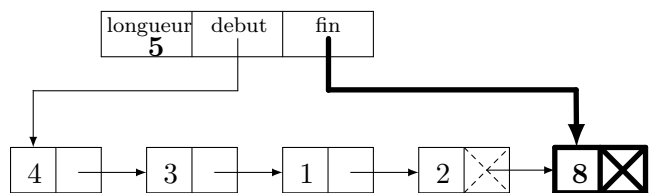


FIGURE 6 - Insertion de 8.

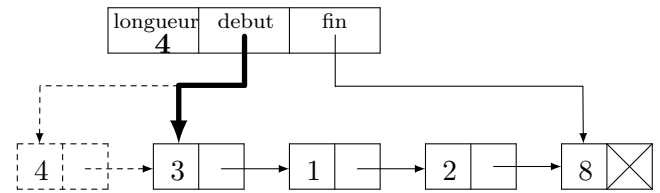


FIGURE 7 - Extraction (on renvoie 4).

12. Ecrire une fonction `maillon* creer_maillon(int v)` renvoyant un (pointeur sur un) nouveau maillon dont le champ `donnee` est fixé à `v` et le champ `suivant` à `NULL`.
13. Ecrire une fonction `file* creer_file(void)` renvoyant (un pointeur sur) une file vide.
14. Ecrire une fonction `int longueur_file(file* f)` renvoyant la longueur de la file `f`.
15. Ecrire une fonction `void ajouter_file(file* f, int v)` qui ajoute un élément à une file. On veillera à gérer correctement l'insertion dans une file vide.

16. Ecrire une fonction `int extraire_file(file* f)` qui modifie la file de sorte à en extraire sa tête, qu'on renverra. On vérifiera la licéité de cette opération à l'aide d'une assertion et on prendra garde à la gestion du cas où la file ne contient qu'un seul élément.
17. Ecrire une fonction `void liberer_maillon(maillon* m)` qui libère la mémoire occupée par tous les maillons suivant le maillon `m`, y compris ce dernier. En déduire une fonction `void liberer_file(file* f)` qui libère la mémoire associée à une file.
18. Pourrait-on efficacement insérer un élément en tête d'une file avec cette structure ? Et supprimer un élément en queue de file ?

On se dote à présent d'une structure permettant de représenter un graphe de flot. **Dans les représentations machines d'un tel graphe, on considèrera toujours que l'arc  $(v, u)$  existe dès que l'arc  $(u, v)$  existe quitte à considérer que  $c(v, u) = 0$ .**

```
struct graphe_flot{
    int n;
    int s;
    int t;
    int **voisins;
    int *degres;
    int **capacites;
};
typedef struct graphe_flot graphe_flot;
```

Les interprétations des champs et les conventions utilisées sont les suivantes :

- L'ensemble des sommets du graphe de flot est  $\llbracket 0, n - 1 \rrbracket$ .
- Les champs `s` et `t` représentent le sommet source et le sommet puits respectivement.
- `degres` est un tableau de taille `n` tel que `degres[u]` est égal au degré sortant du sommet `u`.
- `voisins` est un tableau de taille `n` dont les éléments sont des tableaux. Pour tout  $u \in \llbracket 0, n - 1 \rrbracket$ , le tableau `voisins[u]` est de taille `degres[u]` et contient les voisins du sommet `u`. La remarque en gras précédente implique que si `v` est présent dans `voisins[u]` alors `u` est présent dans `voisins[v]`.
- `capacites` est une matrice de taille  $n \times n$  telle que pour tous sommets  $u, v$ , la case  $(u, v)$  de `capacites` contient  $c(u, v)$ .

Par ailleurs, un flot sera représenté par une matrice d'entiers. On pourra toujours supposer sans le vérifier que, si une fonction prend en entrée un graphe de flot  $G$  et un flot  $f$  représenté par un objet de type `int**` alors  $f$  aura la bonne dimension ; autrement dit, sera de taille  $n \times n$  où  $n$  est le nombre de sommets de  $G$ .

19. Ecrire une fonction `int** zeros(int n)` prenant en entrée un entier  $n$  strictement positif et renvoyant une "matrice" de taille  $n \times n$  remplie de zéros correctement allouée sur le tas.
20. Ecrire une fonction `void liberer_matrice(int** m, int n)` permettant de libérer la mémoire allouée sur le tas par une matrice du type celle allouée par la fonction `zeros`.
21. Ecrire une fonction `int* BFS_residuel(graphe_flot* g, int**f)` prenant en entrée un graphe de flot  $G$  à  $n$  sommets et un flot  $f$  sur  $G$ . Cette fonction devra effectuer un parcours en largeur depuis `s` dans le graphe résiduel  $G_f$  et renvoyer un pointeur `parents` vers un bloc alloué sur le tas de taille  $n$  codant l'arbre de parcours comme suit :

- `parents[g->s]` vaut `g->s`.
  - Pour tout sommet `u` différent de `g->s` visité lors du parcours, `parents[u]` vaut `v` où `v` est le sommet depuis lequel on a exploré `u`.
  - Pour les sommets `u` qui n'ont pas été visités, `parents[u]` vaut `-1`.
22. Ecrire une fonction `int capacite_chemin(graphe_flot* g, int** f, int* parents)` qui prend en entrée un graphe  $G$ , un flot  $f$  et un arbre de parcours `parents` tel que renvoyé par `BFS_residuel` et qui renvoie la capacité disponible sur le chemin reliant  $s$  à  $t$  dans l'arbre `parents`. Si un tel chemin n'existe pas, on renverra 0.
  23. Ecrire une fonction `bool saturer_chemin(graphe_flot* g, int** f, int* parents)` prenant en entrée un graphe de flot  $G$ , un flot  $f$  et un arbre de parcours `parents` ayant pour effet de modifier  $f$  de sorte à saturer l'éventuel chemin entre  $s$  et  $t$  dans l'arbre `parents`. Cette fonction renverra `true` s'il y avait effectivement un chemin entre  $s$  et  $t$  et `false` sinon.
  24. Ecrire une fonction `bool etape(graphe_flot* g, int** f)` prenant en entrée un graphe de flot  $G$ , un flot  $f$  et ayant le comportement suivant :
    - S'il n'existe pas de chemin améliorant pour  $f$ , la fonction renvoie `false`.
    - Sinon, la fonction détermine un chemin améliorant, modifie  $f$  en saturant ce chemin et renvoie `true`.
  25. Déterminer la complexité de la fonction `BFS_residuel` puis celle de `etape` en fonction du nombre de sommets  $|S|$  et du nombre d'arêtes  $|A|$  de  $G$ .
  26. Dédire des questions précédentes une fonction `int** ford_fulkerson(graphe_flot* g)` prenant en entrée un graphe de flot  $G$  et renvoyant le flot  $f$  calculé par l'algorithme de Ford-Fulkerson.
  27. Dans cette question, on suppose que les capacités du graphe  $G$  sont entières et qu'on ne fait aucune hypothèse sur le parcours utilisé pour trouver un chemin améliorant si ce n'est que sa complexité est en  $O(|S| + |A|)$ . Montrer que l'algorithme de Ford-Fulkerson termine et estimer sa complexité en fonction de  $|S|$ ,  $|A|$  et  $M$  où  $M$  est le débit d'un flot maximal sur  $G$ .

*Remarque : dans ce sujet, on n'a pas montré que l'algorithme de Ford-Fulkerson renvoie effectivement un flot maximal. C'est bien le cas et le DS4\* permet de le montrer.*